0808.65688 PATENT APPLICATION

11040 U.S. PTO
09/903208
07/11/01

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re U.S. Patent Application	)
	)
Applicant: Nigel Peter Topham	)
	)
Serial No.	)
	)
Filed: July 11, 2001	)
	)
For: INSTRUCTION SETS FOR PROCESSORS	)
	)
Art Unit:	)

I hereby certify that this paper is being deposited with the United States Postal Service as EXPRESS MAIL in an envelope addressed to: Assistant Commissioner for Patents, Washington, D.C. 20231, on ***July 11, 2001***
Express Label No.: EL 846221610US
Signature: [illegible]

CLAIM FOR PRIORITY

Assistant Commissioner for Patents
Washington, DC 20231

Sir:

Applicant claims foreign priority benefits under 35 U.S.C. § 119 on the basis of the foreign application identified below:

United Kingdom Patent Application No. 0024723.9, filed October 9, 2000.

A certified copy of the priority document is enclosed.

Respectfully submitted,

GREER, BURNS & CRAIN, LTD.

By [illegible]
Patrick G. Burns
Reg. No. 29,367

July 11, 2001
300 South Wacker Drive
Suite 2500
Chicago, IL 60606
(312) 360-0080
Customer Number: **24978**

0808.65688
(312) 360-0080

The Patent Office

PATENTS • DESIGNS • COPYRIGHT • TRADE MARKS

CHARTER MARK

INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

J1040 U.S. PTO
09/903208
07/11/01

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

I also certify that the attached copy of the request for grant of a Patent (Form 1/77) bears an amendment, effected by this office, following a request by the applicant and agreed to by the Comptroller-General.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed [illegible]

Dated 14 MAY 2001

Patents Form 7/77

Patents Act 1977
(Rule 15)

The Patent Office

7/77

09 OCT 2000

Statement of inventorship and of right to grant of a patent

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference	HL76343/000/MRB	
2. Patent application number *(if you know it)*		09 OCT 2000
3. Full name of the or of each applicant	SIROYAN LIMITED	0024723.9

4. Title of the invention

Instruction sets for processors

5. State how the applicant*(s)* derived the right from the inventor*(s)* to be granted a patent

by virtue of the inventor's contract of employment with the applicant

6. How many, if any, additional Patent Forms 7/77 are attached to this form? *(See note (c))*	None	
7.	I/We believe that the person(s) named over the page *(and on any extra copies of this form)* is/are the inventor(s) of the invention which the above patent application relates to. Signature [illegible]	Date 9 October 2000
8. Name and daytime telephone number of person to contact in the United Kingdom	Dr M R Brewer	[020] 7420 0500

Notes

a) If you need help to fill in this form or you have any questions, please contact the patent office on 0645 500505.

b) Write your answers in capital letters using black ink or you may type them.

c) If there are more than three inventors, please write the names and addresses of the other inventors on the back of another Patents Form 7/77 and attach it to this form.

d) When an application does not declare any priority, or declares priority from an earlier UK application, you must provide enough copies of this form so that the patent office can send one to each inventor who is not an applicant.

e) Once you have filled in the form you must remember to sign and date it.

Enter the full names, addresses and postcodes of the inventors in the boxes and underline the surnames

Dr Nigel Peter Topham

~~6 Carolina Close~~
~~Finchampstead,~~
~~Wokingham RG2 6UB~~
~~United Kingdom~~

6 CAROLINA PLACE
FINCHAMPSTEAD
WOKINGHAM
RG40 4PQ
UK

~~7830797002~~

AB
AIC
28/11/0

Patents ADP number *(if you know it)*

07830797002.

Patents ADP number *(if you know it)*

Patents ADP number *(if you know it)*

Reminder:
Have you signed this form?

Patents Form 1/77

Patents Act 1977
(Rule 16)

The Patent Office

10OCT00 E574652-1 [illegible]847
P01/7700 0.00-0024[illegible]3.9

1/77

Request for grant of a patent

(See the notes on the back of this form. You can also get an explanatory leaflet from the Patent Office to help you fill in this form)

09 OCT 2000

The Patent Office

Cardiff Road
Newport
Gwent NP9 1RH

1. Your reference	HL76343/000/MRB	
2. Patent application number *(The Patent Office will fill in this part)*	09 OCT 2000	**0024723.9**
3. Full name, address and postcode of the or of each applicant *(underline all surnames)*	SIROYAN LIMITED 12-15 Hanger ~~Green~~ Lane London ~~W5~~ 3AY ~~United~~ Kingdom	12 - 15 HANGER GREENLAN LONDON W5 3AY UNITED KINGDOM
Patents ADP number *(if you know it)*	AA A/C 28/11/00	7830763003
If the applicant is a corporate body, give the country/state of its incorporation	England	
4. Title of the invention Instruction sets for processors		
5. Full name of your agent *(if you have one)*	Haseltine Lake & Co.	
"Address for service" in the United Kingdom to which all correspondence should be sent *(including the postcode)*	Imperial House 15-19 Kingsway London WC2B 6UD	
Patents ADP number *(if you know it)*	34001	

6. If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and *(if you know it)* the or each application number	Country	Priority application number *(if you know it)*	Date of filing *(day/month/year)*
7. If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date of the earlier application	Number of earlier application		Date of filing *(day/month/year)*
8. Is a statement of inventorship and of right to a grant of patent required in support of this request? *(Answer "Yes" if:* *a) any applicant named in part 3 is not an inventor, or* *b) there is an inventor who is not named as an applicant, or* *c) any named applicant is a corporate body.* *See note (d))*	Yes		

9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form	
Description	29 ✓
Claim*(s)*	10 ✓
Abstract	1 ✓
Drawing*(s)*	~~16~~ 16 ✓

10. If you are also filing any of the following,

Priority documents	
Translations of priority documents	
Statement of inventorship and right to a grant of patent *(Patents Form 7/77*	2 ✓
Request for preliminary examination and search *Patents Form 9/77)*	1 ✓
Request for substantive examination *(Patents Form 10/77)*	1 ✓
Any other documents *(please specify*	Letter requesting accelerated prosecution ✓

11. I/We request the grant of a patent on the basis of this application

Signature [illegible] Date 9 October 2000

12. Name and daytime telephone number of person to contact in the United Kingdom — Dr M R Brewer — [020] 7420 0500

Warning
After an application for a patent has been filed, the Comptroller of the Patent Office will consider whether publication or communication of the invention should be prohibited or restricted under Section 22 of the Patents Act 1977. You will be informed if it is necessary to prohibit or restrict your invention in this way. Furthermore, if you live in the United Kingdom, Section 23 of the Patents Act 1977 stops you from applying for a patent abroad without first getting written permission from the Patent Office unless an application has been filed at least 6 weeks beforehand in the United Kingdom for a patent for the same invention and either no direction prohibiting publication or communication has been given, or any such direction has been revoked.

Notes
a) If you need help to fill in this form or you have any questions, please contact the Patent Office on 0645 500505.
b) Write your answers in capital letters using black ink or you may type them.
c) If there is not enough space for all the relevant details on any part of this form, please continue on a separate sheet of paper and write "see continuation sheet" in the relevant part(s). Any continuation sheet should be attached to this form.
d) If you have answered "Yes" Patents Form 7/77 will need to be filed.
e) Once you have filled in the form you must remember to sign and date it.

DUPLICATE

INSTRUCTION SETS FOR PROCESSORS

The present invention relates to instruction sets for processors. In particular, the present invention relates to processors having two or more different instruction sets. The present invention also relates to methods of automatically encoding instructions for such processors.

A high-performance processor is generally required to have an instruction set which can meet two requirements: compact code (so that the amount of memory required to store the processor's program is desirably small), and a rich set of operations and operands. Such requirements are particularly important in the case of an embedded processor, i.e. a processor embedded in a system such as in a mobile communications device. In this case, high code or instruction density is of critical importance because of the limited resources of the system, for example in terms of available program memory.

However, these two requirements tend to conflict with one another and are difficult to achieve in a single unified instruction set, as compact code involves a minimal encoding for each of the most frequent operations (eliminating the less frequent operations from the instruction set) whereas a rich set of operations and operands requires an orthogonal 32-bit reduced instruction set. Consequently, in a processor having a pre-existing 32-bit instruction set it has been proposed to add a compact 16-bit instruction set which provides the most commonly-used functions and/or access to a limited subset of register operands.

Fig. 1 of the accompanying drawings shows schematically the instruction sets in such a processor. Internally, at the hardware level, the processor has a set of 32-bit instructions IS_{INT}. Externally, the processor has two instruction sets IS_1 and IS_2. The first instruction set IS_1 is made up of the same 32-bit instructions as the internal instruction set IS_{INT}. The second instruction IS_2 is made up of 16-bit instructions and the processor contains instruction translation circuitry 200 for translating each 16-bit instruction of the external instruction set IS_2 into a corresponding one of the 32-bit instructions of the internal instruction set IS_{INT}.

An embedded processor may also be a very long instruction word (VLIW) processor capable of executing VLIW instructions. The most important additional feature of a VLIW processor is Instruction-Level Parallelism (ISP), i.e. its ability to issue two or more operations simultaneously when executing VLIW instructions.

In such a VLIW processor an instruction issuing unit has a plurality of issue slots, each connected operatively to a different execution unit. It is typical for a VLIW processor that issues two or more instructions per processing cycle to encode each instruction in a different format (or group of formats) depending on the issue slot from which the instruction will be issued. The instructions that will be issued in the same processing cycle are combined together in a VLIW packet or parcel. The position of an instruction in the VLIW parcel determines the sub-set of formats in which that instruction may be encoded. In this way, formats for instructions destined for different positions within the VLIW parcel can use identical encodings without introducing ambiguity.

In practice, empirical observation suggests that 90% or more of the instructions within a program are executed so infrequently that they make up 10% or less of the execution time. Naturally, the remaining 10% of the instructions occupy 90% of the execution time. Furthermore, it is often the case that the infrequently-executed parts of a program will not be able to make effective use of the processor's ability to issue two or more instructions simultaneously. If such parts of the program were encoded using a VLIW instruction set, a large proportion of the instructions would be "no operation" (NOP) instructions inserted in the program by the compiler simply to pad out the VLIW parcels when consecutive instructions cannot appear in the same VLIW parcel because the result of one instruction is used by the next. It follows that, for parts of a program where no effective advantage can be taken of the ability to issue instructions in parallel, or where any performance gain from that ability will have little impact anyway, it is desirable to encode the program to achieve maximum code density (i.e. using the smallest possible number of bits).

Accordingly, it is desirable to provide a VLIW processor with a compact-format instruction set, so as to combine the instruction-level parallelism of VLIW architecture with the compact code "footprint" of a tightly-encoded instruction set such as a 16-bit instruction set.

In the previously-proposed processor discussed above with reference to Fig. 1, the compact instruction set was added after the design of an original 32-bit instruction set, with the result that the translation from the 16-bit instructions into 32-bit instructions is undesirably complex and slow.

It is therefore also desirable to design the instruction-set formats and encodings in such a way that the translation from each external instruction format (e.g. at least one VLIW format, and at least one compact format) into a form that can be executed directly by hardware, can be achieved more efficiently.

According to a first aspect of the present invention there is provided a processor having: respective first and second external instruction formats in which instructions are received by the processor, each instruction having an opcode which specifies an operation to be executed, and each external format having one or more preselected opcode bits in which the opcode appears; an internal instruction format into which instructions in the external formats are translated prior to execution of the operations; wherein: the operations include a first operation specifiable in both said first and second external formats, and a second operation specifiable in said second external format; said first and second operations have distinct opcodes in said second external format; and in each said preselected opcode bit which the first and second external formats have in common, the opcodes of the first operation in the two external formats are identical.

According to a second aspect of the present invention there are provided processor instruction encodings having: respective first and second external instruction formats in which the instructions are received by a processor, each instruction having an opcode which specifies an operation to be executed, and each external format having one or more preselected opcode bits in which the opcode appears; an internal instruction format into which the processor instructions in the external formats are translated prior to execution of the operations; wherein: a first operation executable by the processor is specifiable in both said first and second external formats, and a second operation executable by the processor is specifiable in said second external format; said first and second operations have distinct opcodes in said second external format; and in each said preselected opcode bit which the first and second external formats have in common, the opcodes of the first operation in the two external formats are identical.

According to a third aspect of the present invention there is provided a method of encoding processor instructions for a processor having respective first and second external instruction formats in which instructions are received by the processor, each instruction having an opcode which specifies an operation to be executed, and each external format having one or more preselected opcode bits in which the opcode appears, the processor also having an internal instruction format into which instructions in the external formats are translated prior to execution of the operations, and the operations include a first operation specifiable in both said first and second external formats, and a second operation specifiable in said second external format, said method comprising the steps of: encoding said first and second operations with distinct opcodes in said second external format; and encoding the opcodes of the first operation in said first and second external formats so that, in each said preselected opcode bit which the first and second external formats have in common, the opcodes of the first operation in the two external formats are identical.

According to a fourth aspect of the present invention there is provided a method of encoding instructions for a processor having two or more external instruction formats and one or more internal instruction formats, the method comprising: a) selecting initial encoding parameters including a number of effective opcode bits in each external and internal format and a set of mapping functions, each said mapping function serving to translate an opcode specified by the said opcode bits in one of the external formats to an opcode specified by the said opcode bits in the, or in one of the, internal formats; (b) allocating each operation executable by the processor an opcode distinct from that allocated to each other operation in each external and internal format in which the operation is specifiable, the allocated opcodes being such that each relevant mapping function translates such an external-format opcode allocated to the operation into such an internal-format opcode allocated to the operation and such that all the internal-format opcodes allocated to the operation have the same effective opcode bits; and c) if in step (b) no opcode is available for allocation in each specifiable format for every one of the said operations, determining which of the said encoding parameters is constraining the allocation in step (b), relaxing the constraining parameter, and then repeating step (b).

According to a fifth aspect of the present invention there is provided a computer program which, when executed, encodes instructions for a processor having two or more external instruction formats and one or more internal instruction formats, the computer program comprising code portions for: (a) selecting initial encoding parameters including a number of effective opcode bits in each external and internal format and a set of mapping functions, each said mapping function serving to translate an opcode specified by the said opcode bits in one of the external formats to an opcode specified by the said opcode bits in the, or in one of the, internal formats; (b) allocating each operation executable by the processor an opcode distinct from that allocated to each other operation in each external and internal format in which the operation is specifiable, the allocated opcodes being such that each relevant mapping function translates such an external-format opcode allocated to the operation into such an internal-format opcode allocated to the operation and such that all the internal-format opcodes allocated to the operation have the same effective opcode bits; and (c) if in step (b) no opcode is available for allocation in each specifiable format for every one of the said operations, determining which of the said encoding parameters is constraining the allocation in step (b), relaxing the constraining parameter, and then repeating step (b).

Reference will now be made, by way of example, to the accompanying drawings, in which:

Fig. 1, discussed hereinbefore, is a schematic diagram for use in explaining a previously-proposed processor having an additional compact instruction set;

Fig. 2 shows parts of a processor embodying the present invention;

Fig. 3(A) shows a schematic diagram for use in explaining previously-considered instruction encodings;

Fig. 3(B) shows a schematic diagram corresponding to Fig. 3(A) for use in explaining congruent instruction encodings;

Figs. 4(A) and 4(B) present a flowchart for use in explaining a method of encoding instructions embodying the present invention;

Fig. 5 shows a schematic view of external and internal instruction formats in a specific example;

Fig. 6 presents a table illustrating which operations are specifiable in each external and internal format in the Fig. 5 specific example;

Figs. 7(A) to 7(H) present schematic diagrams for use in explaining different stages of an automatic encoding method applied to the Fig. 5 specific example; and

Fig. 8 shows the final instruction encodings achieved by the method of Fig. 7.

Fig. 2 shows parts of a processor embodying the present invention. In this example, the processor is a very long instruction word (VLIW) processor. The processor 1 includes an instruction issuing unit 10, a schedule storage unit 12, respective first, second and third VLIW translation units 4, 6 and 8, a scalar translation unit 9, respective first, second and third execution units 14, 16 and 18, and a register file 20.

The instruction issuing unit 10 has three issue slots IS1, IS2 and IS3 connected respectively to the first, second and third translation units 4, 6 and 8. Respective outputs of the first, second and third translation units 4, 6 and 8 are connected to respective first inputs of the first, second and third execution units 14, 16 and 18 respectively.

The instruction issuing unit 10 has a further output SC connected to the scalar translation unit 9. An output of the scalar translation unit 9 is connected in common to a second input of each execution unit 14, 16 and 18.

A first bus 22 connects all three execution unit 14, 16 and 18 to the register file 20. A second bus 24 connects the first and second units 14 and 16 (but not the third execution unit 18 in this embodiment) to a memory 26 which, in this example, is an external random access memory (RAM) device. The memory 26 could alternatively be a RAM internal to the processor 1.

Incidentally, although Fig. 1 shows shared buses 22 and 24 connecting the execution units to the register file 20 and memory 26, it will be appreciated that alternatively each execution unit could have its own independent connection to the register file and memory.

The processor 1 performs a series of processing cycles. The processor may operate selectively in two modes: a scalar mode and a VLIW mode.

In scalar mode the processor executes instructions from a particular instruction set (which may or may not be distinct from the VLIW instruction set). In this mode instructions are not issued at the issue slots IS1 to IS3.

In VLIW mode, on the other hand, the instruction issuing unit 10 can issue up to 3 instructions in parallel per cycle at the 3 issue slots IS1 to IS3, i.e. the full instruction issue width is exploited.

Scalar-mode instructions and VLIW-mode instructions are both stored together in the schedule storage unit 12. The instructions are issued according to an instruction schedule stored in the schedule storage unit.

As explained later in more detail, instructions in the instruction schedule are written in at least two different external formats, including at least one format belonging to a scalar instruction set of the processor (hereinafter a "scalar format") and at least one format belonging to a VLIW instruction set of the processor (hereinafter a "VLIW format"). In practice, there may be two or more scalar formats and two or more VLIW formats. In the case of the VLIW formats it is possible to have different formats for different issue slots, although a format may be shared by two or more issue slots.

On the other hand, within the processor each execution unit executes instructions in at least one internal format. Accordingly, each execution unit 14, 16 and 18 is provided with a translation unit 4, 6 or 8 which translates an instruction in one of the external VLIW formats into the (or, if more than one, the appropriate) internal format required by the execution unit concerned. Similarly, the scalar translation unit 9 is provided for translating an instruction in one of the external scalar formats into the (appropriate) internal format required by the execution units.

After translation by the relevant translation unit 4, 6, 8 or 9 the instructions issued by the instructing issuing unit 10 at the different issue slots or at the scalar instruction output SC are executed by the corresponding execution units 14, 16 and 18. Each of the execution units may be designed to execute more than one instruction at the same time, so that execution of a new instruction can be initiated prior to completion of execution of a previous instruction issued to the execution unit concerned.

To execute instructions, each execution unit 14, 16 and 18 has access to the register file 20 via the first bus 22. Values held in registers contained in the register file 20 can therefore be read and written by the execution units 14, 16 and 18. Also, the first and second execution units 14 and 16 have access via the second bus 24 to the external memory 26 so as to enable values stored in memory locations of the external memory 26 to be read and written as well. The third execution unit 18 does not have access to the external memory 26 and so can only manipulate values contained in the register file 20 in this embodiment.

As outlined above, the architecture of the Fig. 2 processor defines a compact (e.g. 16-bit) instruction set and a wider (e.g. 32-bit) VLIW instruction set. There are at least two of these wider instructions in each VLIW parcel. Instructions belonging to the compact instruction set and the VLIW instruction set are encoded using external formats.

There is also at least one internal instruction format to which all instructions in an external format are translated during execution.

Each VLIW parcel is made up of two or more instructions at different positions (slots) within the parcel. Each slot within a VLIW parcel may contain an instruction encoded in one of several external VLIW formats. At least some fundamental operations provided by the processor (e.g. add, subtract or multiply) may need to be available in two or more, or possibly all, of the instruction slots of a VLIW parcel. In this case, the same fundamental operation may be encoded in a different external format per instruction slot. Of course, when the instructions in these different external formats are translated they must all have the same operation code (opcode) within the same group of bits in the or each internal format.

A fundamental operation may also need to be available using two or more scalar instructions, for example where the same fundamental operation is performed using two or more different types of operand or operand addressing. In this case, each of the two or more scalar instructions relating to the same fundamental operation must be encoded using a different scalar format and must translate to a different internal format. Again, when translated into an internal format, these two or more scalar instructions must have the same opcode as all VLIW-format instructions for the same operation which translate to the same internal format. Typically, the scalar instruction set will be a sub-set of the full (VLIW) instruction set, allowing a more compact encoding of the external scalar formats.

The task of designing formats and assigning codes to each operation in each format is complicated by the fact an operation X may appear in external formats F_1 and F_2, whereas another operation Y may appear in the external format F_2 and in a further external format F_3. This means that the design of the external formats F_1, F_2 and F_3, and the choice of opcodes for operations X and Y, are interdependent. Fig. 3(A) shows a simple example of previously-considered instruction encodings. In this example, an add operation appears both in external formats F_1 and F_2. The add operation in both formats F_1 and F_2 is mapped to the same internal format G_1. A load instruction appears in the external format F_2 and in the further external format F_3. The load operation in both formats is translated into the same internal format G_2.

As shown in Fig. 3(A), in the different external formats F_1 to F_3, different sets of bits are used for specifying the opcode, i.e. the opcode fields are different. In the format F_1 the four bits from bit i+1 to bit i+4 are used to specify the opcode. In format F_2, the three bits from bit i+1 to bit i+3 are used to specify the opcode. In format F_3, the four bits from bit i to i+3 are used to specify the opcode. The opcode field for F_2 may be shorter than for F_1 and F_3 because there are less operations available in F_2, for example.

In Fig. 3(A) the external formats F_1 and F_2 have the bits i+1 to i+3 in common as opcode bits. For the add operation in format F_1 and the load operation in format F_2 these common bits i+1 to i+3 are the same, even though the operations are different. This complicates the translation process. For example, in internal format G_1 the add operation may have the opcode "1011". The add operation in format F_2 can be translated into this internal-format opcode simply by selecting "101" from F_2 and appending a "1". However, to translate the add operation in format F_1 into this internal-format code it is not possible to use a simple selection operation. In this case it may be necessary to examine all opcode bits i+1 to i+4 in the external format F_1 and match uniquely the pattern of bits ("1101") which identifies the add operation in format F_1. Anything short of this full examination might not distinguish it from another operation in F_1.

However, if it could be guaranteed that:

(i) the opcodes for "add" and "load" in format F_2 are distinct, and the same is true for any other pair of operations which appear together in the same format F_2 as well as in at least one other format; and

(ii) every operation that appears in two or more external formats (i.e. the "add" operation and any other which appears in F_1 and F_2, and the "load" operation and any other which appears in F_2 and F_3) is identically coded in all common opcode bits in all those formats in which it appears;

then the translation process can be independent of the opcodes themselves and can rely only on discovering the external format (and, if there is more than one internal format, the target internal format) of each instruction. Instruction encodings which have this property are referred to herein as "congruent" instruction encodings.

In Fig. 3(B) the add and load operations of Fig. 3(A) have been allocated congruent instruction encodings. It can be observed that the opcodes assigned to the add instruction ("1011" in format F_1 and "101" in format F_2) are identical in the three opcode bits that are in common for the two formats F_1 and F_2 ("101").

Similarly, in the case of the load operation appearing in formats F_2 and F_3, the three opcode bits that are in common for formats F_2 and F_3 are identical ("011") in F_2 and F_3.

Thus, the instruction encodings in Fig. 3 are congruent. This means that the translation operation performed by the translation unit can be a simple bit-selection operation, for example to select some or all of the bits from i+1 to i+4 in the case of translation from external format F_1 to internal format G_1, selecting some or all of the three bits from i+1 to i+3 in the case of translation from external format F_2 to either internal format G_1 or G_2, and selecting some or all of the four bits from i to i+3 when translating from external format F_3 to internal format G_2. The particular selection of bits required for a given translation can then be determined simply by identifying the external format and target internal format. The identification of the external format can be made by examining ID bits in the external formats, for example the bits labelled F_1 to F_3 in Fig. 3(B).

The task of designing instruction formats and opcodes having the property of congruence is not difficult in the simple case illustrated in Fig. 3(B) in which only two operations are considered. However, when there are many operations in different external formats which also appear in different internal formats the task of designing formats and assigning opcodes becomes very difficult. For example, a processor may have approximately 32 to 128 instructions in its scalar instruction set, 32 to 128 (or possibly double that) instructions in its VLIW instruction set, and perhaps 3 to 6 different external formats and 4 to 6 different internal formats.

This has meant that heretofore the translation units used to carry out the translations have been undesirably complex, leading to propagation delays and excessive power consumption in previously-considered processors.

Next, a method will be described for designing automatically formats, opcodes and translations for achieving congruent instruction encodings.

In order to describe this method for determining opcode fields within instruction formats and deriving congruent encodings in those formats let us begin by defining the terms we shall use.

Let

$$W = \bigcup_{j=1}^{N} G_j$$

be the set of all internal instructions, encoded in N internal formats G_j.

Each internal format G_j is a proper subset of W, and comprises a set of internal instructions defined by the processor that is being implemented. If y is an instruction encoded in format G_j, then the opcode for y is given by function $g_j(y)$ which selects a sub-field containing a_j bits from the instruction format G_j.

Let F_i denote an external instruction format, where $i \in [1, M]$. If x is an instruction encoded in format F_i, then the code for x is given by the function $f_i(x)$ which selects a sub-field containing b_i bits from the instruction F_i.

Each internal instruction is represented in memory by one or more external instruction formats. Where an instruction is represented in two or more external formats, each variant must translate to the same internal opcode. These variants typically perform the same function, though the types and representation of their operands may differ.

The present explanation is concerned with the process by which opcode field widths are determined, and the process by which operation codes are assigned in each format. The encoding of operands is also important, but is independent of the issue of opcode assignment and is therefore not addressed here.

A translation from external format F_i to internal format G_j requires a mapping function $m_{i,j}$ which maps the b_i bits of opcode from F_i to the a_j bits of opcode in G_j. For the purposes of simplicity in implementation and tractability in design the mappings are preferably bit selections or permutations. In this explanation it will also be assumed that there is only one mapping function for translating between any pair of external and internal formats.

The instruction set architecture of the processor defines for each internal instruction y an associated set of translations, T_y, where each translation is a pair $\langle i, j\rangle$ identifying an external format as the source of the translation and an internal format as the destination of the translation. For each translation there must exist a mapping function $m_{i,j}$. Hence:

$$T_y = \{\langle i,j\rangle : (y \in G_j) \wedge (x \in F_i) \wedge (y = m_{i,j}(x))\}$$

... (eq 1)

Each format, whether internal or external, has a cardinality determined by the number of opcodes within the format. The cardinality of F_i is written $|F_i|$, and hence the sizes of the opcode fields in external and internal formats must satisfy the following inequalities:

$$a_j \geq \log_2(|G_j|)$$
$$b_i \geq \log_2(|F_i|)$$

... (eq 2)

Each internal format G_j therefore defines opcodes in the range $|0, 2^{aj}\rangle$, and each external format F_i defines opcodes in the range $|0, 2^{bi}\rangle$. At any point during the method Q_j contains the set of opcodes available to be allocated to operations in internal format G_j. Similarly, R_i contains the set of opcodes available to be allocated to operations in external format F_i.

The problem now consists of determining an unique opcode for each instruction $y \in N$, and determining suitable selection- or permutation-based mapping functions for each translation defined in the instruction set architecture. One preferred embodiment of the method can now be expressed in pseudo-code, using the terminology introduced above, as shown in the flowchart of Figs. 4(A) and 4(B).

Each mapping function $m_{i,j}$ initially maps a chosen number b_i of effective opcode bits of the external format F_i to a chosen number a_j of effective opcode bits of the internal format G_j. This can map no more than $q = min(a_j, b_i)$ bits from external format F_i to a_j bits in internal format G_j, setting any undefined bits in a_j to zero. For simplicity, it will be assumed in this preferred embodiment that each mapping function involves selecting all bits of the external-format opcode to be some or all of the bits of the internal-format opcode after translation. Other mapping functions can be used in other embodiments of the invention, for example mapping functions involving permutations.

The method begins in step S1 by first computing the minimum possible number a_j or b_i of opcode bits that could theoretically encode the number of instructions in each external format and each internal format. This minimum possible number a_j or b_i is used as an initial number of effective opcode bits for the format concerned.

In step S2, a new series of iterations is started (as explained later, several series may be required in a practical situation). Firstly, for each internal format G_j, a set Q_j of available opcodes is formed, made up of all possible opcodes definable by the a_j bits. Similarly, for each external format F_i, a set R_i of available opcodes is assigned, made up of all possible opcodes definable by the b_i bits. As explained later, each available opcode may have a working number of bits greater than the computed minimum possible number a_j or b_i of opcode bits. For example, the working number for all available opcodes in all sets Q_j and R_i may be set equal to the highest computed minimum possible number a_j or b_i.

Step S3 involves iterating through all operations in the internal formats and determining their opcodes in each external format where they occur.

During each series of iterations, steps S4 to S9 are performed per iteration. One fundamental operation is considered per iteration. In step S4, for the considered operation, the method examines the pair of sets R_i and Q_j for the external format and internal format of each mapping function needed to translate the considered operation, and identifies as a mutual set h_t any members the two sets of the pair have in common. In step S5 a set H of common members of all the mutual sets h_t for all the needed mapping functions is formed. If the result is an empty set in step S6, then no allowable mapping is found and the method goes to step S11 where the constraints are relaxed. If H contains at least one common opcode, step S7 selects the or one of the common opcodes in H.

Then in step S8 the selected opcode is removed from each set R_i and Q_j for the external and internal formats in which the considered operation appears, i.e. the sets examined in step S4.

The method terminates when it is determined in step S9 that the method has successfully allocated opcodes to all operations in all the required external and internal formats.

The method is guaranteed to terminate because the back-tracking process in step S11 successively relaxes the encoding constraints until there are as many opcode bits as are needed to find a congruent assignment of codes.

In addition to selecting bits from the external format F_i, the mapping function may also permute the bits. For example, the order of the bits may be reversed by the mapping function. Such permutations can be used when the number of mapped bits reaches q, where $q=\min(a_j, b_i)$.

If $p = max(a_j, b_i)$, then the total number of possible permutations is $p!/(p-q)!$. Hence, for large instruction sets, the number of possible permutations could be very large. In practice, however, it is typical for p to be about 5 and q to be about 3. This means a maximum of 60 different permutation functions for each mapping. Typically one might expect there to be five different mappings, leading to a total of 60^5 possible sets of mapping functions to consider on each iteration of the method defined by steps S4 to S9 (*i.e.* 778 million possibilities). This is within the capabilities of a modern computer to enumerate and evaluate automatically.

For larger field widths the number of possible permutations grows intractably large. However, it is still possible to operate the method successfully in this case by restricting the class of permutations that will be searched. For example, there are $n(n+1)/2$ possible permutations of n-bit field defined by swapping arbitrary pairs of bits. By choosing such a restriction on the possible permutations to be examined by the method the running time of the method could be constrained to be polynomial in n.

Next, operation of the method described with reference to Figs. 4(A) and 4(B) will be illustrated with reference to a specific example. In this example, a VLIW processor, for example a processor generally in accordance with Fig. 2, has the capability to issue two instructions simultaneously from issue slots A and B respectively.

Referring to Fig. 5, it can be seen that the external VLIW formats allowed for instructions to be issued from issue slot A include first and second external VLIW formats F_1 and F_2. The opcode bits in external format F_1 are denoted by C_1 in Fig. 5, and the opcode bits in format F_2 are denoted by C_2.

In the case of instructions to be issued from issue slot B, two external VLIW formats are also available, one of them is the same external format F_2 as available at issue slot A, and the other is a third external VLIW format F_3. The opcode bits in format F_3 are denoted by C_3 in Fig. 5.

In addition, the processor in this example is capable of operating in a scalar mode to execute instructions in one of two different 16-bit scalar external formats F_4 and F_5. The opcode bits in format F_4 are denoted by C_4 in Fig. 5, and the opcode bits in format F_5 are denoted by C_5.

The processor in this example also has two internal formats G_1 and G_2. The opcode bits in the internal format G_1 are denoted by C_A in Fig. 5, and the opcode bits in internal format G_2 are denoted by C_B. Each scalar instruction translates into a single operation in one or both of the internal formats G_1 and G_2, encoded in either the C_A or C_B field.

As also shown schematically in Fig. 5, the processor has three translation units, 30, 32 and 34. The translation unit 30 corresponds to issue slot A and is operable to translate opcode bits C_1 in external format F_1 or opcode bits C_2 in external format F_2 into either opcode bits C_A in internal format G_1 or opcode bits C_B in internal format G_2.

Similarly, the translation unit 32 corresponds to issue slot B and is operable to translate opcode bits C_2 in external format F_2 or opcode bits C_3 in external format F_3 into opcode bits C_A in internal format G_1 or opcode bits C_B in internal format G_2.

The translation unit 34 corresponds to the scalar instructions and is operable to translate either opcode bits C_4 in external format F_4 or opcode bits C_5 in external format F_5 into opcode bits C_A in internal format G_1 or opcode bits C_B in internal format G_2.

It will be appreciated that the translation units 30 and 32 in Fig. 5 correspond to the translation units 4, 6 and 8 in Fig. 2, and that the translation unit 34 in Fig. 5 corresponds to the translation unit 9 in Fig. 2.

Referring now to Fig. 6, the processor in the present example has a small set of seven fundamental operations: an addition operation add, a logical OR operation or, a multiply operation mul, a load immediate operation li, a subtraction operation sub, a return from VLIW-mode operation rv and a division operation div. The table presented in Fig. 6 lists these seven fundamental operations in the first (left-hand) column. The second column in Fig. 6 indicates in which internal formats the operation concerned is permitted to appear. The add, or, mul, li and sub instructions are permitted to appear in both internal formats G_1 and G_2 and so have "G1" and "G2" rows, but the rv and div instructions are only permitted to appear in internal format G_2 and so have no "G1" row.

The remaining six columns in Fig. 6 relate to the five external instruction formats F_1 to F_5. The external format F_2 has two columns allocated to it in this case, as this format is allowed at both issue slot A and issue slot B.

Each cell in one of the six external-format columns corresponds to an instruction. Some of the cells are shaded whilst others are blank. An instruction I in a cell at row G_j and F_i must be represented in external format F_i and must be translated to internal format G_j if its cell is shaded. If the cell is not shaded then the instruction I concerned is not present in external format F_i. Take, for example, the cell denoted by an asterisk in Fig. 6. This cell is at row G_1 for the or instruction, and at column F_1. The shading of the cell indicates that the or instruction is present in external format F_1 and internal format G_1, requiring that opcodes for the or operation are appropriately chosen in both formats and that a translation exists for the or instruction between these two formats.

The algorithm described previously with reference to Figs. 4(A) and 4(B) will now be applied to the present example of Figs. 5 and 6 to determine the opcodes, the opcode field widths in each format, and the mapping functions (translations) between formats.

The set W of fundamental operations in this example can be written as:

$$W = \{add, or, mul, li, sub, rv, div\}$$

... (eq 3)

The number N of internal formats is 2 (G_1 and G_2), and the number M of external formats is 5 (F_1 to F_5).

Looking at Fig. 6, for each external format F_i a mapping function $m_{i,j}$ is required if, for any operation, there is a shaded cell in row G_j. For example, taking the external format F_1, it can be seen that a mapping function is required for internal format G_1 but not for internal format G_2, as no cell in the F_1 column is shaded in a G_2 row.

Thus, the following mapping functions are required in the present example: $m_{1,1}$, $m_{2,1}$, $m_{2,2}$, $m_{3,2}$, $m_{4,1}$, $m_{4,2}$, $m_{5,1}$ and $m_{5,2}$.

The translation pairs t for each operation, which are derived directly from Fig. 6, are as follows:

$$\begin{bmatrix} T_{add}=\{\langle 1,1\rangle,\langle 2,1\rangle,\langle 2,2\rangle,\langle 3,2\rangle,\langle 4,1\rangle,\langle 4,2\rangle,\langle 5,1\rangle,\langle 5,2\rangle\} \\ T_{or}=\{\langle 1,1\rangle,\langle 2,1\rangle,\langle 2,2\rangle,\langle 3,2\rangle,\langle 4,1\rangle,\langle 4,2\rangle,\langle 5,1\rangle,\langle 5,2\rangle\} \\ T_{mul}=\{\langle 1,1\rangle,\langle 2,1\rangle,\langle 2,2\rangle,\langle 3,2\rangle,\langle 4,1\rangle,\langle 4,2\rangle,\langle 5,1\rangle,\langle 5,2\rangle\} \\ T_{li}=\{\langle 2,1\rangle,\langle 2,2\rangle,\langle 3,2\rangle,\langle 4,1\rangle,\langle 4,2\rangle\} \\ T_{sub}=\{\langle 1,1\rangle,\langle 3,2\rangle,\langle 5,1\rangle,\langle 5,2\rangle\} \\ T_{rv}=\{\langle 3,2\rangle\} \\ T_{div}=\{\langle 3,2\rangle,\langle 5,2\rangle\} \end{bmatrix}$$

... (eq 4)

In step S1 of the algorithm (Fig. 4(A)) the number of opcodes required in each format is determined. For each external format this is determined by observing the number of operations for which there is at least one shaded cell in the column for that external format. For example, in the case of the external format F_1 it can be seen that four operations (add, or, mul and sub) have a shaded cell in the column concerned. Where an external format has two columns (such as the external format F_2) an operation is only counted once even if it appears in one internal format in one column and internal format in another column. Thus, in the case of the external format F_2, the number of operations $|F_2|$ is 4.

In the case of an internal format the number of opcodes required is calculated by counting the total number of rows (containing at least one shaded cell) allocated to the internal format concerned. For example, the internal format G_1 has five rows with shaded cells. The internal format G_2 has seven rows with shaded cells.

Thus, the numbers of opcodes required in the different internal and external formats are: $|G_1|=5$, $|G_2|=7$, $|F_1|=4$, $|F_2|=4$, $|F_3|=6$, $|F_4|=4$ and $|F_5|=5$.

As a result, in step S1, the initial numbers of effective opcode bits are determined as $a_1=3$, $a_2=3$, $b_1=2$, $b_2=2$, $b_3=3$, $b_4=2$ and $b_5=3$. These numbers represent the minimum possible numbers of bits that could theoretically encode the number of operations appearing in the format concerned, and may have to be increased in the course of execution of the algorithm.

In step S2, a set of available opcodes is created for each external format and for each internal format, as shown in equation 5.

$$\begin{aligned}
R_1 &= \{000,\ 001,\ 010,\ 011\} \\
R_2 &= \{000,\ 001,\ 010,\ 011\} \\
R_3 &= \{000,\ 001,\ 010,\ 011,\ 100,\ 101,\ 110,\ 111\} \\
R_4 &= \{000,\ 001,\ 010,\ 011\} \\
R_5 &= \{000,\ 001,\ 010,\ 011,\ 100,\ 101,\ 110,\ 111\} \\
Q_1 &= \{000,\ 001,\ 010,\ 011,\ 100,\ 101,\ 110,\ 111\} \\
Q_2 &= \{000,\ 001,\ 010,\ 011,\ 100,\ 101,\ 110,\ 111\}
\end{aligned} \qquad \text{...(eq 5)}$$

The working number of bits in each opcode is initially set to be equal to the highest required number of opcode bits amongst any of the internal and external formats, i.e. 3 opcode bits as required by the formats G_1, G_2 and F_5. The initial set R_1 of opcodes for external format F_1 is made up of four three-bit codes 000, 001, 010 and 011. Four codes are required as b_1 was calculated to be 2 in step S1. The same is true for the other two-bit external formats R_2 and R_4.

In the case of the external formats F_3 and F_5 eight codes are required and the initial codes assigned to R_3 and R_5 are 000, 001, 010, 011, 100, 101, 110 and 111.

Each of the internal formats G_1 and G_2 also requires eight codes ($a_1=3$ and $a_2=3$) so the initial sets Q_1 and Q_2 of opcodes for these internal formats are also the same as for the external formats R_3 and R_5.

In step S3 a first series of iterations is commenced, and in this first series the first operation in Fig. 6, i.e. the add operation, is selected for initial consideration.

In step S4, the available opcodes for the operation that are unused (not yet allocated) in each relevant pair of external and internal formats (8 pairs in all: F_1-G_1, F_2-G_1, F_4-G_1, F_5-G_1, F_2-G_2, F_3-G_2, F_4-G_2, F_5-G_2 in this case) are considered. Because no opcodes have yet been allocated, for the 5 pairs F_1-G_1, F_2-G_1, F_4-G_1, F_2-G_2 and F_4-G_2 h_t = {000, 001, 010, 011} while for the 3 pairs F_5-G_1, F_3-G_2 and F_5-G_2 h_t = {000, 001, 010, 011, 100, 101, 110, 111}. Thus, in step S5 H={000, 001, 010, 011}.

In step S6 it is checked whether H is empty. In this case it is not, so processing proceeds to step S7. Here, the opcode c=000 is selected first from H. The opcode 000 therefore becomes allocated to the add operation.

In step S8 the internal-format opcode sets Q_1 and Q_2 are updated to remove therefrom the opcode 000, if contained therein. Thus, the code 000 is removed from each of the sets Q_1 and Q_2.

Also in step S8 the set of available opcodes for each relevant external format (in this case all of the external formats F_1 to F_5) is updated to remove therefrom the opcode 000, if contained therein. Thus, 000 is removed from each of the sets R_1 to R_5.

The results of the allocations performed in the first iteration are shown in Fig. 7(A). In Figs. 7(A) to 7(H) the opcodes remaining in the sets Q or R are shown. Also, any opcode allocations made in the external and internal formats are entered in the relevant cells.

Processing then returns to step S3 for the second iteration of this series. In the second iteration, the or operation is considered. The pairs to be considered in step S4 are the same as for the first iteration. The results of steps S4 and S5 are that H={001, 010, 011}. Thus, in step S6, H is not empty and processing proceeds to step S7. In step S7 the opcode c=001 is selected. Accordingly, in step S8, the opcode 001 is removed from each of the sets Q_1 and Q_2 of available opcodes for the internal formats G_1 and G_2. Similarly, in the sets R_1 to R_5 for the external formats F_1 to F_5, the code 001 is removed. The results after the second iteration are shown in Fig. 7(B).

In the third iteration, the mul operation is considered. Again, the pairs to be considered in step S4 are the same as for the first and second iterations. In this case, the result H of the computation performed in step S5 is {010, 011}, so that, in step S7, the opcode 010 is selected. In step S8 the opcode 010 is removed from all the sets Q_1 to Q_2 and R_1 to R_5.

Thus, 010 becomes allocated to the mul operation. Fig. 7(C) shows the state reached at this time.

In the fourth iteration of the series the li instruction is considered. In this case the pairs to be examined in step S4 are F_2-G_1, F_4-G_1, F_2-G_2 and F_4-G_2. In step S5 of this iteration it is determined that H={011}. As the H set is not empty, processing goes on to step S7. Here, the code 011 is selected (it is the only code available in the set H). The code 011 therefore becomes assigned to li. This code is removed from the relevant sets Q_1, Q_2, R_2 and R_4, but is left in the sets R_1, R_3 and R_5. The resulting state is shown in

Fig. 7(D).

In the fifth iteration, the sub instruction is considered. In step S4 the set of translations T_{sub} = {<1,1>, <3,2>, <5,1>, <5,2>}. Accordingly, as the pairs of external and internal formats for these translations are F_1-G_1, F_5-G_1, F_3-G_2, F_5-G_2 the common sets h_t are {} for F_1-G_1 and {100, 101, 110, 111} for F_5-G_1, F_3-G_2 and F_5-G_2.

This means H={⌀} in step S5. This is because, although 100, 101, 101, 110 and 111 are still unused in R_3, R_5, Q_1 and Q_2, none of these codes is available in the remaining relevant set R_1 which only contains 011. Accordingly, processing proceeds via step S6 to step S11 in which the constraint is assessed. It is determined that the intersection between R_1 and Q_1 (and between R_1 and Q_2) is the empty set. Since R_1 has less members than Q_1 and Q_2 it can reasonably be concluded that R_1 is the constraining factor. To overcome this constraint the number of effective opcode bits for F_1 needs to be increased beyond its initial value of 2. Accordingly, a_1 is increased by one to 3. The remaining values a_2 to a_5, b_1 and b_2 are left unchanged.

Now, all of the existing opcode assignments are void and a second series of iterations is commenced at step S2. In this series of iterations R_1 = {000, 001. 010, 011, 100, 101, 110, 111} initially. In the fifth iteration of this second series the sub instruction is again considered. At this stage the sate is shown in Fig. 7(E).

This time, in step S5 H={100, 101, 110, 111}. In step S7 the opcode 100 is selected. In step S8, 100 is removed from R_1, R_3, R_5, Q_1 and Q_2. The resulting state is shown in Fig. 7(F).

In the sixth iteration of the second series, the rv instruction is considered for the first time. In step S5 H={101, 110, 111}. In step S7 the opcode 101 is selected. In step S8, 101 is removed from R_3 and Q_2. The resulting state is shown in Fig. 7(G).

In the seventh iteration of the second series, the div instruction is considered for the first time. In step S5 H={110, 111}. In step S7 the opcode 110 is selected. In step S8, 100 is removed from R_3, R_5 and Q_2, The resulting state is shown in Fig. 7(H).

At this point all instructions have been allocated opcodes and the processing moves to step S10. In this step the opcodes assigned so far are examined to determine how many bits in each external format actually need to be provided in the instructions in the external format concerned. For example, in the external format F_4 all the allocated codes 000, 001, 010 and 011 have the prefix 0. This means that the prefix 0 is entirely redundant is external format F_4. Accordingly, provided that the format F_4 can still be distinguished from all other external formats, the prefix 0 can be omitted from instructions in format F_4 so that only a 2-bit opcode field is required for format F_4. The same is true for external format F_2.

It follows of course that the mapping functions $m_{4,1}$, $m_{4,2}$, $m_{2,1}$ and $m_{2,2}$ must insert the 0 prefix during translation so that the add, or mul and li operations in format F4 are distinguished from the sub, rv and div operations in formats F_1, F_3 and F_5.

This optimisation step S10 becomes particularly important when the number of prefix bits is greater than the number of bits in each instruction set needed to give each operation a distinct opcode in each external format.

The final opcodes after optimisation are shown in Fig. 8.

A method embodying the present invention can be implemented by a general-purpose computer operating in accordance with a computer program. This computer program may be carried by an suitable carrier medium such as a storage medium (e.g. floppy disk or CD Rom) or a signal. Such a carrier signal could be a signal downloaded via a communications network such as the Internet. The appended computer program claims are to be interpreted as covering a computer program by itself or in any of the above-mentioned forms.

Although the above description relates, by way of example, to a VLIW processor it will be appreciated that the present invention is applicable to processors other than VLIW processors. A processor embodying eh present invention may be included as a processor "core" in a highly-integrated "system-on-a-chip" (SOC) for use in multimedia applications, network routers, video mobile phones, intelligent automobiles, digital television, voice recognition, 3D games, etc.

CLAIMS:

1. A processor having:

respective first and second external instruction formats in which instructions are received by the processor, each instruction having an opcode which specifies an operation to be executed, and each external format having one or more preselected opcode bits in which the opcode appears;

an internal instruction format into which instructions in the external formats are translated prior to execution of the operations;

wherein:

the operations include a first operation specifiable in both said first and second external formats, and a second operation specifiable in said second external format;

said first and second operations have distinct opcodes in said second external format; and

in each said preselected opcode bit which the first and second external formats have in common, the opcodes of the first operation in the two external formats are identical.

2. A processor as claimed in claim 1, wherein:

the operations include one or more further first operations, each specifiable in both said first and second external formats, and one or more further second operations specifiable in said second external format;

for every pair of operations, made up of one said first operation and one said second operation, the operations of the pair have distinct opcodes in said second external format; and

in each said preselected opcode bit which the first and second external formats have in common, the opcodes of each first operation in the two external formats are identical.

3. A processor as claimed in claim 1 or 2, having:

a third external instruction format in which instructions are received by the processor, each instruction having an opcode which specifies an operation to be executed, and said third external format having one or more preselected opcode bits in which the opcode appears;

respective first and second internal instruction formats into which instructions in the external formats are translated prior to execution of the operations;

wherein:

said second operation is specifiable in both said second and third external formats;

an instruction specifying said first operation in either said first or second external format is translated into said first internal format, and an instruction specifying said second operation in either said second or third external format is translated into said second internal format; and

in each said preselected opcode bit which the second and third external formats have in common, the opcodes of the second operation in the two external formats are identical.

4. A processor as claimed in claim 3, wherein:

the operations include one or more further first operations, each specifiable in both said first and second external formats, and one or more further second operations specifiable in said second external format;

for every pair of operations, made up of one said first operation and one said second operation, the operations of the pair have distinct opcodes in said second external format;

in each said preselected opcode bit which the first and second external formats have in common, the opcodes of each first operation in the two external formats are identical; and

in each said preselected opcode bit which the second and third external formats have in common, the opcodes of each first operation in the two external formats are identical.

5. A processor as claimed in any preceding claim, being a VLIW processor, wherein one external format is a scalar instruction format used for scalar instructions, and another external format is a VLIW instruction format used for VLIW instructions.

6. A processor as claimed in any preceding claim, being a VLIW processor, wherein the external formats are or include two different VLIW formats.

7. A processor as claimed in claim 6, wherein the two different VLIW formats are used in different respective instruction slots of a VLIW instruction parcel.

8. A processor as claimed in claim 6 or 7, wherein at least one instruction slot of a VLIW instruction parcel uses the two different VLIW formats.

9. A processor as claimed in any preceding claim, wherein one external format has an instruction width different from that of another external format.

10. A processor as claimed in any preceding claim, having:

translation means operable to perform a predetermined translation operation for translating each said external-format opcode into a corresponding internal-format opcode.

11. A processor as claimed in claim 10, wherein said translation operation involves selecting and/or permuting bits amongst the said preselected opcode bits in the external-format instruction.

12. A processor as claimed in claim 10 or 11, wherein the translation operation is independent of the external-format opcode.

13. A processor as claimed in claim 12, wherein the translation means are operable to identify the internal format into which each external-format instruction is to be translated, and to carry out the said translation operation according to the identified internal format.

14. Processor instruction encodings having:

respective first and second external instruction formats in which the instructions are received by a processor, each instruction having an opcode which specifies an operation to be executed, and each external format having one or more preselected opcode bits in which the opcode appears;

an internal instruction format into which the processor instructions in the external formats are translated prior to execution of the operations;

wherein:

a first operation executable by the processor is specifiable in both said first and second external formats, and a second operation executable by the processor is specifiable in said second external format;

said first and second operations have distinct opcodes in said second external format; and

in each said preselected opcode bit which the first and second external formats have in common, the opcodes of the first operation in the two external formats are identical.

15. A method of encoding processor instructions for a processor having respective first and second external instruction formats in which instructions are received by the processor, each instruction having an opcode which specifies an operation to be executed, and each external format having one or more preselected opcode bits in which the opcode appears, the processor also having an internal instruction format into which

instructions in the external formats are translated prior to execution of the operations, and the operations include a first operation specifiable in both said first and second external formats, and a second operation specifiable in said second external format, said method comprising the steps of:

encoding said first and second operations with distinct opcodes in said second external format; and

encoding the opcodes of the first operation in said first and second external formats so that, in each said preselected opcode bit which the first and second external formats have in common, the opcodes of the first operation in the two external formats are identical.

16. A method of encoding instructions for a processor having two or more external instruction formats and one or more internal instruction formats, the method comprising:

(a) selecting initial encoding parameters including a number of effective opcode bits in each external and internal format and a set of mapping functions, each said mapping function serving to translate an opcode specified by the said opcode bits in one of the external formats to an opcode specified by the said opcode bits in the, or in one of the, internal formats;

(b) allocating each operation executable by the processor an opcode distinct from that allocated to each other operation in each external and internal format in which the operation is specifiable, the allocated opcodes being such that each relevant mapping function translates such an external-format opcode allocated to the operation into such an internal-format opcode allocated to the operation and such that all the internal-format opcodes allocated to the operation have the same effective opcode bits; and

(c) if in step (b) no opcode is available for allocation in each specifiable format for every one of the said operations, determining which of the said encoding parameters is constraining the allocation in step (b), relaxing the constraining parameter, and then repeating step (b).

17. A method as claimed in claim 16, wherein each said mapping function involves selecting all bits of the external-format opcode as some or all of the bits of the internal-format opcode.

18. A method as claimed in claim 16 or 17, wherein in step (a), for each external and internal format, the said number of effective opcode bits is made equal to a minimum possible number of opcode bits that could theoretically encode the number of operations specifiable in the format concerned.

19. A method as claimed in any one of claims 16 to 18, wherein step (b) comprises a series of iterations, and prior to commencing the series of iterations a set of available opcodes in each external and internal format is formed, and in each iteration of the series one said operation is considered and the allocation of the opcode to the considered operation is made based on an examination of the sets of available opcodes in each external and internal format in which the considered operation is specifiable.

20. A method as claimed in claim 19, wherein, for each said external and internal format, the set of available opcodes formed prior to commencing a series of iterations has a number of members dependent upon the said number of effective opcode bits currently applicable to that format.

21. A method as claimed in claim 19 or 20, wherein the available opcodes in all the sets have the same working number of bits.

22. A method as claimed in claim 21, wherein the

said working number is set equal to a minimum possible number of opcode bits that could theoretically encode the number of operations specifiable in the external or internal format having the highest number of operations specifiable in the format concerned.

23. A method as claimed in any one of claims 19 to 22, wherein each said iteration of step (b) comprises:

(b-1) determining which, if any, available opcodes are common to the sets for all the external and internal formats in which the considered operation is specifiable; and

(b-2) if it is determined in step (b-1) that one or more such available opcodes are common, selecting the or one of the common opcodes, allocating it to the considered operation, and removing the selected opcode from the set for each external and internal format in which the considered operation is specifiable.

24. A method as claimed in claim 23 wherein each said iteration of step (b) further comprises:

(b-3) if it is determined in step (b-1) that no common available opcode is present in the sets for all the external and internal formats in which the considered operation is specifiable, making all existing allocated opcodes void and carrying out step (c).

25. A method as claimed in any one of claims 16 to 24, further comprising:

(d) after all of the operations have been allocated one of the said available opcodes having the said working number of bits, determining for each external format whether that working number is greater than a minimum number of bits needed to provide each operation specifiable in that external format with its own distinct opcode and, if so, restricting the allocated opcodes in that external format to the determined minimum number of bits.

26. A method as claimed in claim 25, wherein step (d) comprises:

(d-1) identifying for each external format a maximum-length common prefix, if any, for all allocated opcodes in the external format concerned; and

(d-2) removing the identified common prefix from all the allocated opcodes in the external format concerned; and

(d-3) adjusting each mapping function that serves to translate an opcode specified by the opcode bits in the external format concerned into an opcode specified by internal-format opcode bits so that the mapping function prepends the identified common prefix to the external-format opcode bits during translation.

27. A method as claimed in any one of claims 16 to 26, wherein in step (c) if it is determined that the number of effective opcode bits in one of the external or internal formats is the constraining parameter, the number of effective opcode bits in that format is increased.

28. A method as claimed in any one of claims 16 to 27, carried out by electronic data processing means.

29. A computer program which, when executed, encodes instructions for a processor having two or more external instruction formats and one or more internal instruction formats, the computer program comprising code portions for:

(a) selecting initial encoding parameters including a number of effective opcode bits in each external and internal format and a set of mapping functions, each said mapping function serving to translate an opcode specified by the said opcode bits in one of the external formats to an opcode specified by the said opcode bits in the, or in one of the, internal formats;

(b) allocating each operation executable by the processor an opcode distinct from that allocated to each other operation in each external and internal format in which the operation is specifiable, the allocated opcodes being such that each relevant mapping function translates such an external-format opcode allocated to the operation into such an internal-format opcode allocated to the operation and such that all the internal-format opcodes allocated to the operation have the same effective opcode bits; and

(c) if in step (b) no opcode is available for allocation in each specifiable format for every one of the said operations, determining which of the said encoding parameters is constraining the allocation in step (b), relaxing the constraining parameter, and then repeating step (b).

30. A computer program which, when run on a computer, causes the computer to carry out the encoding method of any one of claims 16 to 28.

31. A computer program as claimed in claim 29 or 30, carried by a carrier medium.

32. A computer program as claimed in claim 31, wherein the said carrier medium is a storage medium.

33. A computer program as claimed in claim 31, wherein the said carrier medium is a signal.

34. A processor substantially as hereinbefore described with reference to any of Figs. 2 to 8 except 3(A) of the accompanying drawings.

35. Processor instruction encodings substantially as hereinbefore described with reference to Figs. 2 to 8 except 3(A) of the accompanying drawings.

36. A method of encoding processor instructions substantially as hereinbefore described with reference to the accompanying drawings.

37. A computer program which, when run on a computer, causes the computer to carry out an encoding

method substantially as hereinbefore described with reference to the accompanying drawings.

ABSTRACT

INSTRUCTION SETS FOR PROCESSORS

A processor has respective first and second external instruction formats (F_1, F_2) in which instructions (add, load) are received by the processor. Each instruction has an opcode (e.g. 1011) which specifies an operation to be executed. Each external format has one or more preselected opcode bits (F_1: i+1~i+4; F_2:i+1~i+3) in which the opcode appears. The processor also has an internal instruction format (G_1) into which instructions in the external formats are translated prior to execution of the operation.

A first operation (add) is specifiable in both the first and second external formats (F_1, F_2), and a second operation (load) is specifiable in the second external format (F_2). The first and second operations have distinct opcodes (101, 011) in the second external format. In each of the preselected opcode bits which the first and second external formats have in common (i+1~i+3), the opcodes of the first operation (101) in the two external formats are identical.

Such "congruent" instruction encodings can enable a translation process, for translating the external-format opcode into a corresponding internal-format opcode, to be carried out simply and quickly without the need to positively identify each individual external-format opcode.

[Fig. 3(B)]

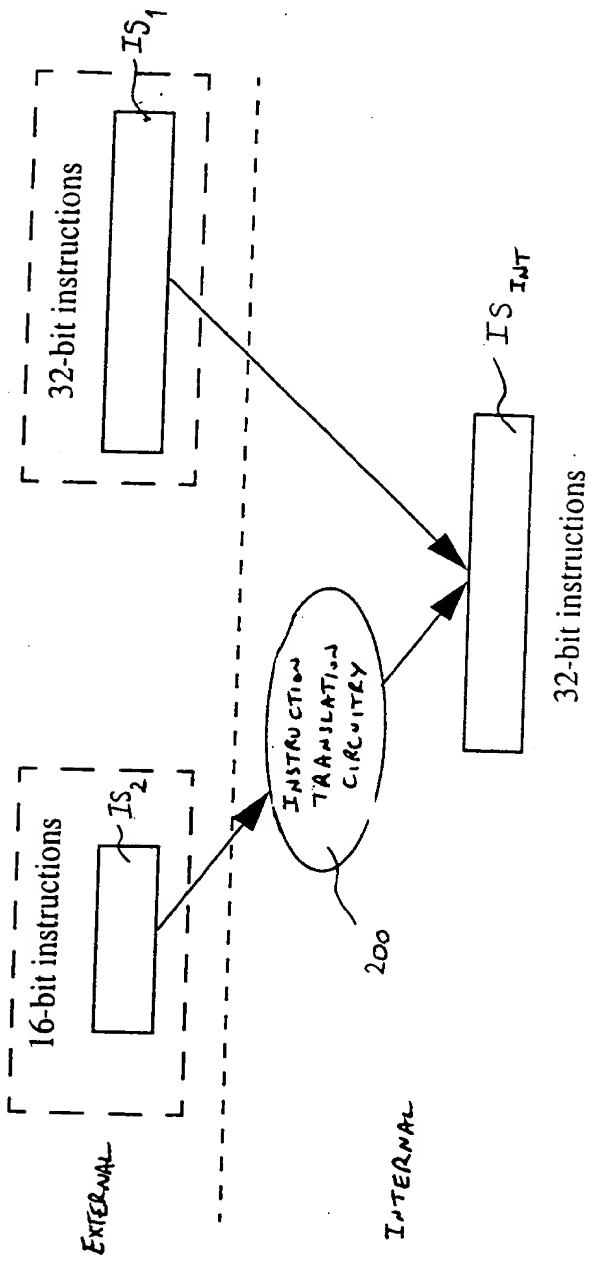

FIGURE 1

NOT TO BE AMENDED

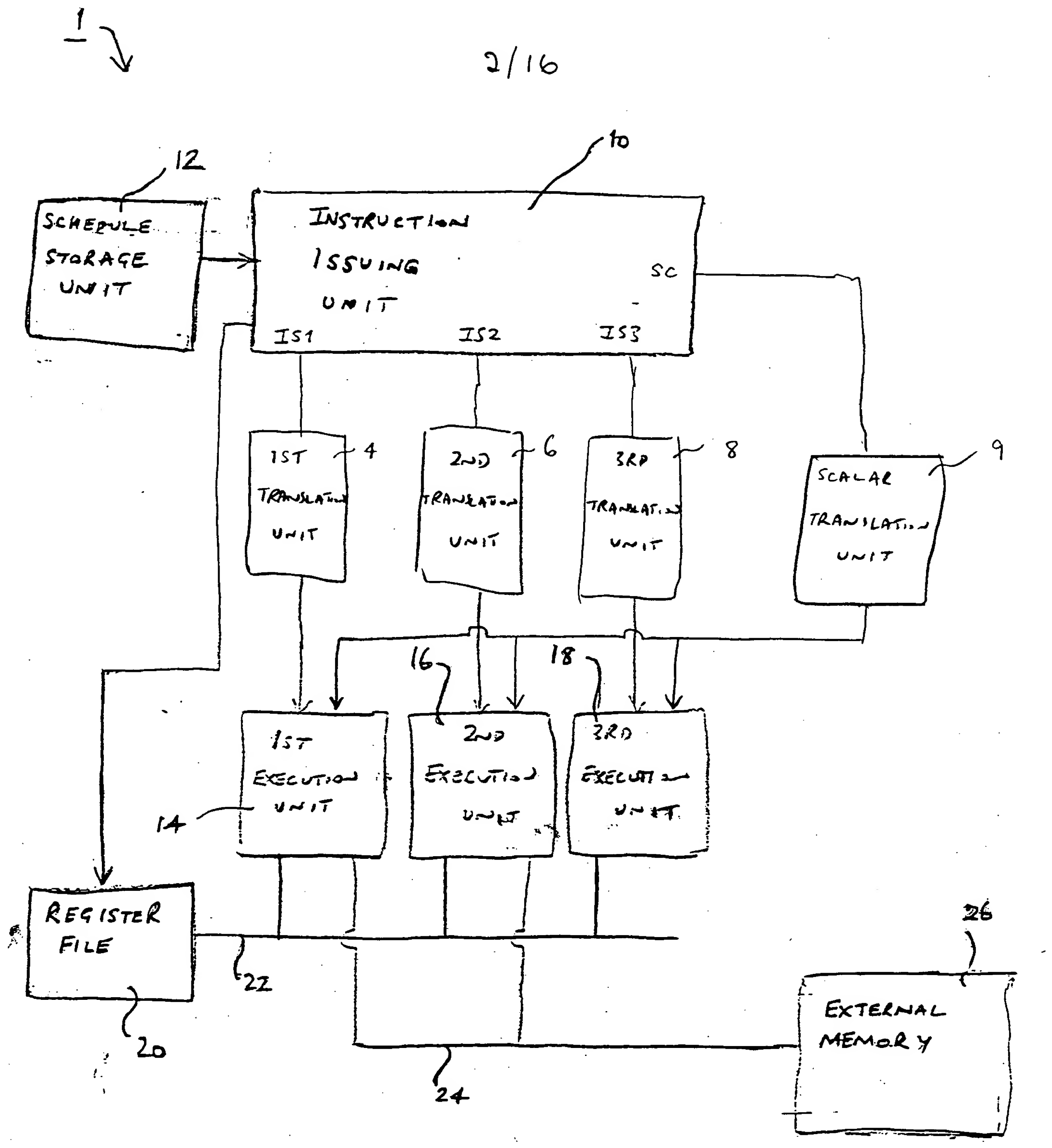

FIGURE 2

NOT TO BE AMENDED

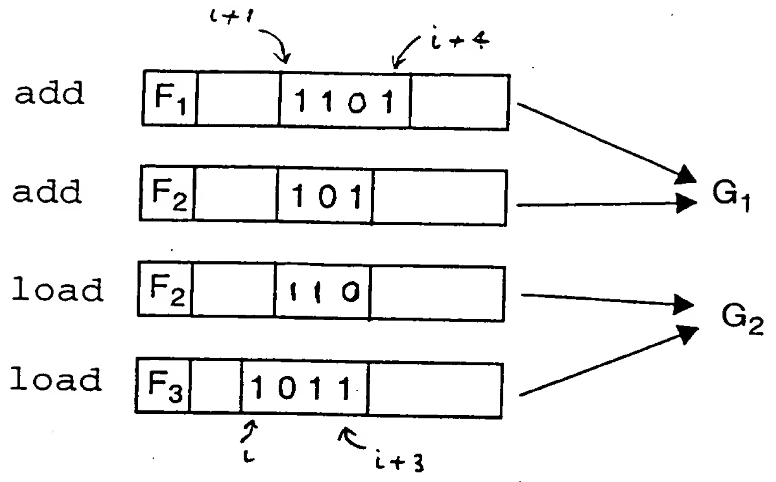

FIGURE 3(A)

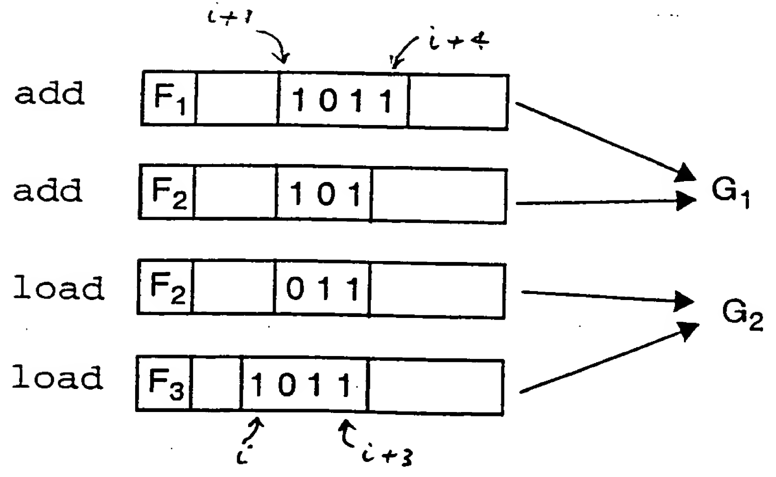

FIGURE 3(B)

NOT TO BE AMENDED

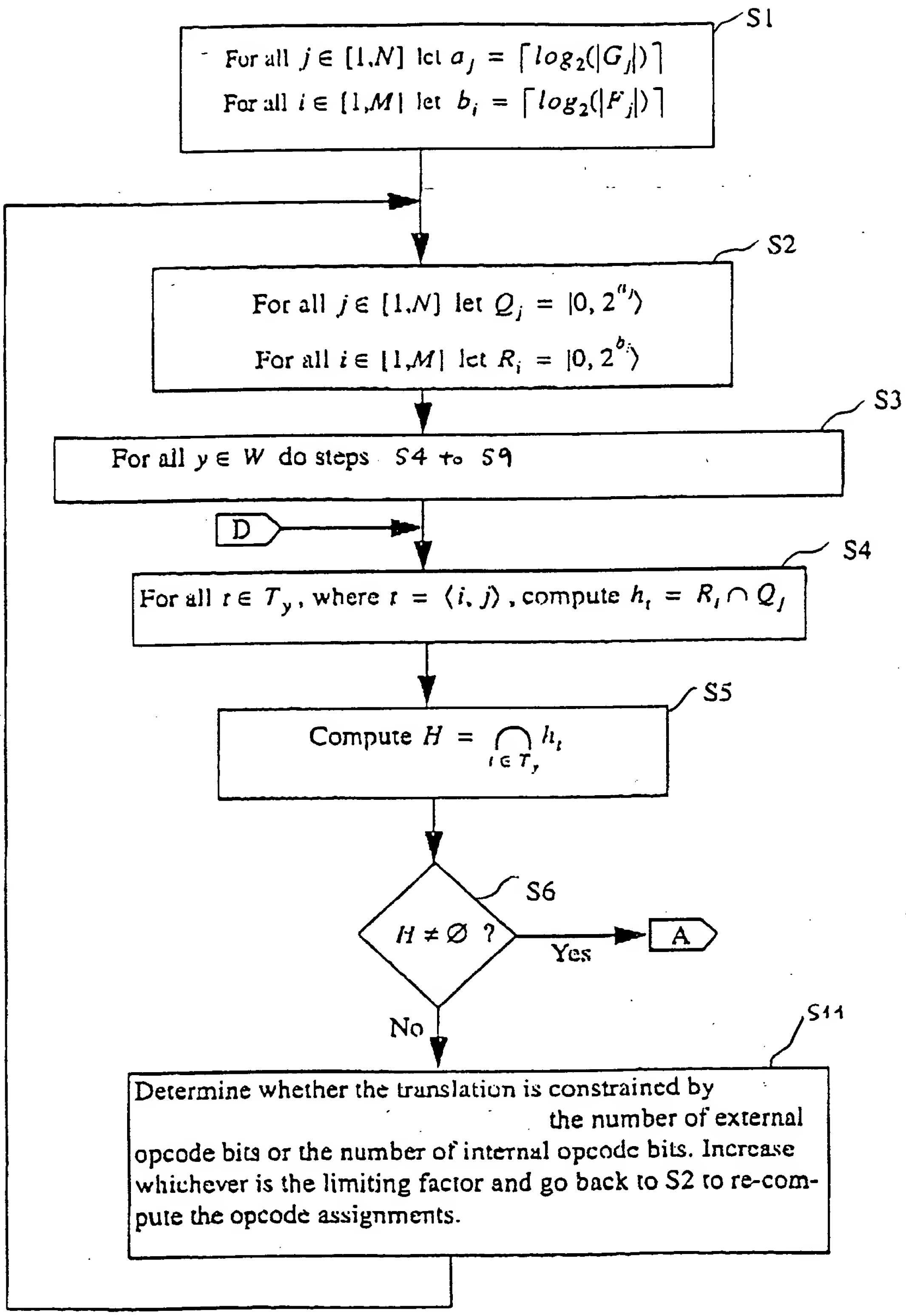
S1
For all $j \in [1,N]$ let $a_j = \lceil log_2(|G_j|) \rceil$
For all $i \in [1,M]$ let $b_i = \lceil log_2(|F_i|) \rceil$
S2
For all $j \in [1,N]$ let $Q_j = [0, 2^{a_j})$
For all $i \in [1,M]$ let $R_i = [0, 2^{b_i})$
S3
For all $y \in W$ do steps S4 to S9
D
S4
For all $t \in T_y$, where $t = \langle i, j \rangle$, compute $h_t = R_i \cap Q_j$
S5
Compute $H = \bigcap_{t \in T_y} h_t$
S6
$H \neq \varnothing$?
Yes
A
No
S11
Determine whether the translation is constrained by the number of external opcode bits or the number of internal opcode bits. Increase whichever is the limiting factor and go back to S2 to re-compute the opcode assignments.

FIG. 4(A)

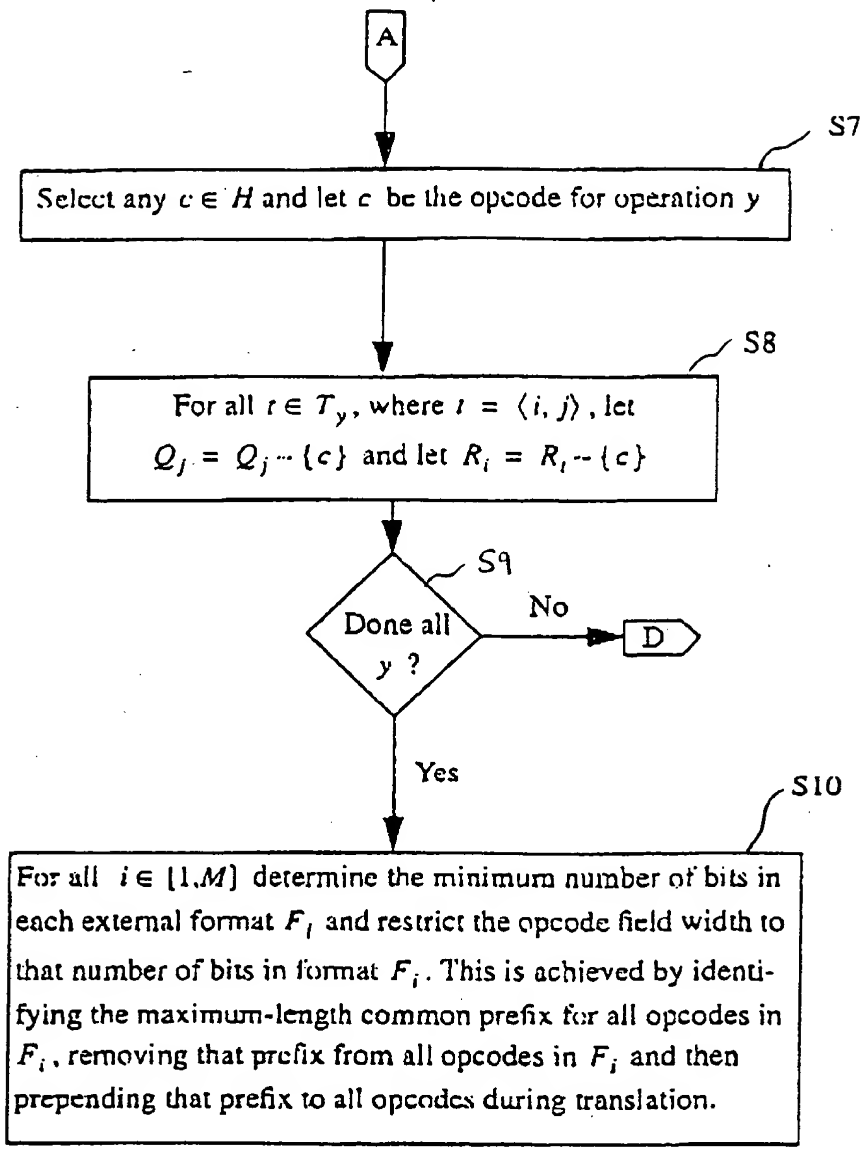
A
S7
Select any $c \in H$ and let c be the opcode for operation y
S8
For all $t \in T_y$, where $t = \langle i, j \rangle$, let $Q_j = Q_j - \{c\}$ and let $R_i = R_i - \{c\}$
S9
Done all y ?
No
D
Yes
S10
For all $i \in [1,M]$ determine the minimum number of bits in each external format F_i and restrict the opcode field width to that number of bits in format F_i. This is achieved by identifying the maximum-length common prefix for all opcodes in F_i, removing that prefix from all opcodes in F_i and then prepending that prefix to all opcodes during translation.

FIG. 4(B)

NOT TO BE AMENDED

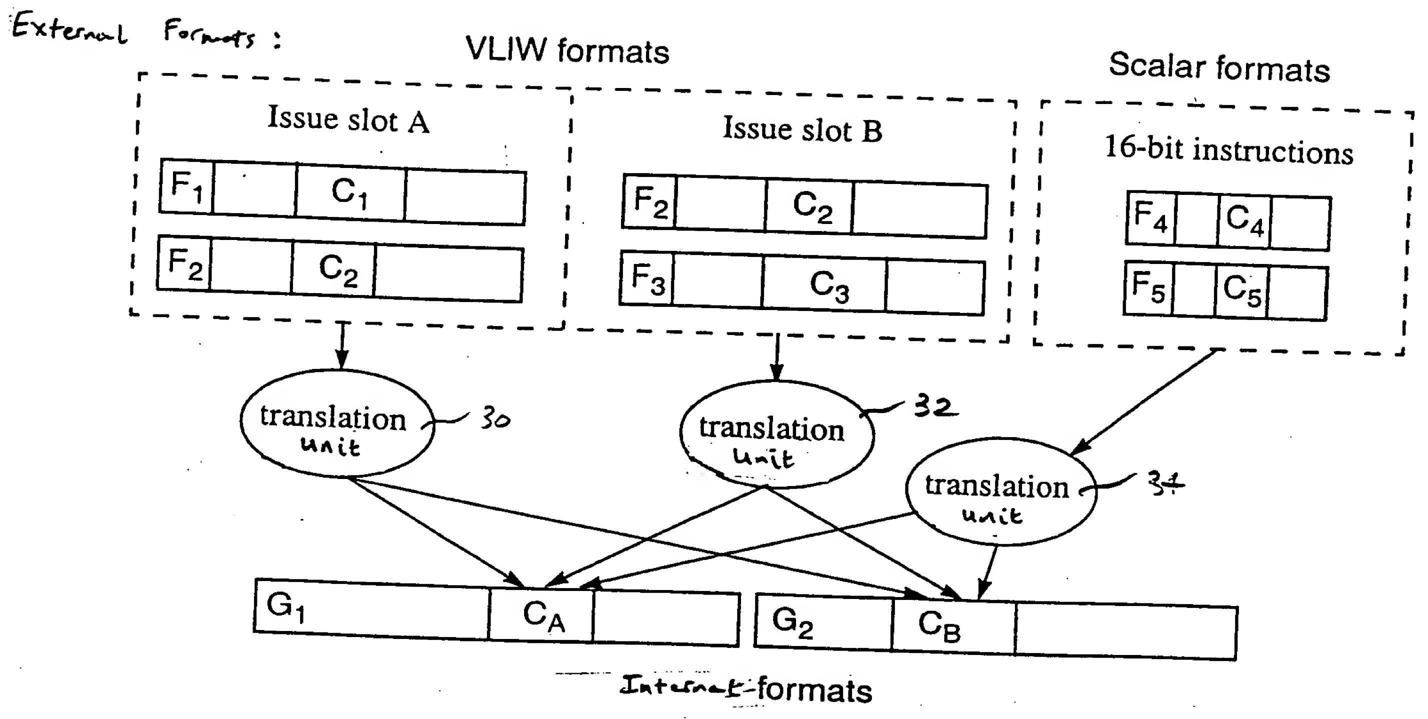
External Formats :
VLIW formats
Scalar formats
Issue slot A
F_1
C_1
F_2
C_2
Issue slot B
F_2
C_2
F_3
C_3
16-bit instructions
F_4
C_4
F_5
C_5
translation unit
30
translation unit
32
translation unit
34
G_1
C_A
G_2
C_B
Internal-formats

FIGURE 5

Operation	Internal' format	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	▨	▨	▨	▨		
	G_2	▨	▨			▨	▨
or	G_1	▨	▨	▨ *	▨		
	G_2	▨	▨			▨	▨
mul	G_1	▨	▨	▨	▨		
	G_2	▨	▨			▨	▨
li	G_1	▨			▨		
	G_2	▨				▨	
sub	G_1		▨	▨			
	G_2		▨				▨
rv	G_2						▨
div	G_2		▨				▨

FIGURE 6

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	000	000	000	000		
	G_2	000	000			000	000
or	G_1						
	G_2						
mul	G_1						
	G_2						
li	G_1						
	G_2						
sub	G_1						
	G_2						
rv	G_2						
div	G_2						

$R_1 = \{001, 010, 011\}$
$R_2 = \{001, 010, 011\}$
$R_3 = \{001, 010, 011, 100, 101, 110, 111\}$
$R_4 = \{001, 010, 011\}$
$R_5 = \{001, 010, 011, 100, 101, 110, 111\}$
$Q_1 = \{001, 010, 011, 100, 101, 110, 111\}$
$Q_2 = \{001, 010, 011, 100, 101, 110, 111\}$

FIG. 7(A)

NOT TO BE AMENDED

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	000	000	000	000		
	G_2	000	000			000	000
or	G_1	001	001	001	001		
	G_2	001	001			001	001
mul	G_1						
	G_2						
li	G_1						
	G_2						
sub	G_1						
	G_2						
rv	G_2						
div	G_2						

$R_1 = \{010, 011\}$

$R_2 = \{010, 011\}$

$R_3 = \{010, 011, 100, 101, 110, 111\}$

$R_4 = \{010, 011\}$

$R_5 = \{010, 011, 100, 101, 110, 111\}$

$Q_1 = \{010, 011, 100, 101, 110, 111\}$

$Q_2 = \{010, 011, 100, 101, 110, 111\}$

FIG. 7(B)

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	000	000	000	000		
	G_2	000	000			000	000
or	G_1	001	001	001	001		
	G_2	001	001			001	001
mul	G_1	010	010	010	010		
	G_2	010	010			010	010
li	G_1						
	G_2						
sub	G_1						
	G_2						
rv	G_2						
div	G_2						

$R_1 = \{011\}$

$R_2 = \{011\}$

$R_3 = \{011, 100, 101, 110, 111\}$

$R_4 = \{011\}$

$R_5 = \{011, 100, 101, 110, 111\}$

$Q_1 = \{011, 100, 101, 110, 111\}$

$Q_2 = \{011, 100, 101, 110, 111\}$

FIG. 7(c)

NOT TO BE AMENDED

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	000	000	000	000		
	G_2	000	000			000	000
or	G_1	001	001	001	001		
	G_2	001	001			001	001
mul	G_1	010	010	010	010		
	G_2	010	010			010	010
li	G_1	011			011		
	G_2	011				011	
sub	G_1						
	G_2						
rv	G_2						
div	G_2						

$R_1 = \{011\}$
$R_2 = \{\ \}$
$R_3 = \{011, 100, 101, 110, 111\}$
$R_4 = \{\ \}$
$R_5 = \{011, 100, 101, 110, 111\}$
$Q_1 = \{100, 101, 110, 111\}$
$Q_2 = \{100, 101, 110, 111\}$

FIG. 7(D)

NOT TO BE AMENDED

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	000	000	000	000		
	G_2	000	000			000	000
or	G_1	001	001	001	001		
	G_2	001	001			001	001
mul	G_1	010	010	010	010		
	G_2	010	010			010	010
li	G_1	011			011		
	G_2	011				011	
sub	G_1						
	G_2						
rv	G_2						
div	G_2						

$R_1 = \{011, 100, 101, 110, 111\}$

$R_2 = \{\ \}$

$R_3 = \{011, 100, 101, 110, 111\}$

$R_4 = \{\ \}$

$R_5 = \{011, 100, 101, 110, 111\}$

$Q_1 = \{100, 101, 110, 111\}$

$Q_2 = \{100, 101, 110, 111\}$

FIG. 7(E)

NOT TO BE AMENDED

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	000	000	000	000		
	G_2	000	000			000	000
or	G_1	001	001	001	001		
	G_2	001	001			001	001
mul	G_1	010	010	010	010		
	G_2	010	010			010	010
li	G_1	011			011		
	G_2	011				011	
sub	G_1		100	100			
	G_2		100				100
rv	G_2						
div	G_2						

$R_1 = \{011, 101, 110, 111\}$
$R_2 = \{\ \}$
$R_3 = \{011, 101, 110, 111\}$
$R_4 = \{\ \}$
$R_5 = \{011, 101, 110, 111\}$
$Q_1 = \{101, 110, 111\}$
$Q_2 = \{101, 110, 111\}$

FIG. 7(F)

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_3	F_1	F_2	F_2	F_3
add	G_1	000	000	000	000		
	G_2	000	000			000	000
or	G_1	001	001	001	001		
	G_2	001	001			001	001
mul	G_1	010	010	010	010		
	G_2	010	010			010	010
li	G_1	011			011		
	G_2	011				011	
sub	G_1		100	100			
	G_2		100				100
rv	G_2						101
div	G_2						

$R_1 = \{011, 101, 110, 111\}$

$R_2 = \{\ \}$

$R_3 = \{011, 110, 111\}$

$R_4 = \{\ \}$

$R_5 = \{011, 101, 110, 111\}$

$Q_1 = \{101, 110, 111\}$

$Q_2 = \{110, 111\}$

FIG. 7(G)

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	000	000	000	000		
	G_2	000	000			000	000
or	G_1	001	001	001	001		
	G_2	001	001			001	001
mul	G_1	010	010	010	010		
	G_2	010	010			010	010
li	G_1	011			011		
	G_2	011				011	
sub	G_1		100	100			
	G_2		100				100
rv	G_2						101
div	G_2		110				110

$R_1 = \{011, 101, 110, 111\}$

$R_2 = \{\ \}$

$R_3 = \{011, 111\}$

$R_4 = \{\ \}$

$R_5 = \{011, 101, 111\}$

$Q_1 = \{101, 110, 111\}$

$Q_2 = \{111\}$

FIG. 7(H)

NOT TO BE AMENDED

Operation	Internal formats	External formats (Scalar)		External formats (VLIW)			
				Issue slot A		Issue slot B	
		F_4	F_5	F_1	F_2	F_2	F_3
add	G_1	00	000	000	00		
	G_2	00	000			00	000
or	G_1	01	001	001	01		
	G_2	01	001			01	001
mul	G_1	10	010	010	10		
	G_2	10	010			10	010
li	G_1	11			11		
	G_2	11				11	
sub	G_1		100	100			
	G_2		100				100
rv	G_2						101
div	G_2		110				110

$R_1 = \{011, 101, 110, 111\}$

$R_2 = \{\ \}$

$R_3 = \{011, 111\}$

$R_4 = \{\ \}$

$R_5 = \{011, 101, 111\}$

$Q_1 = \{101, 110, 111\}$

$Q_2 = \{111\}$

FIG. 8

NOT TO BE AMENDED